

MULTIMEDIA



UNIVERSITY

STUDENT ID NO

--	--	--	--	--	--	--	--	--	--

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2, 2016/2017 SESSION

ECE3206 – OBJECT ORIENTED PROGRAMMING WITH C++ (CE, EE, ME & TE)

02 MARCH 2017
9:00 A.M – 11:00 A.M.
(2 Hours)

INSTRUCTIONS TO STUDENT

1. This question paper consists of eight (8) pages only (including this page).
2. **This is an open-book exam**, so the questions are more in-depth and there are fewer questions than normal exams.
3. There are **THREE (3) QUESTIONS** in this paper. Answer **ALL QUESTIONS**. Questions 1 and 2 carry 35 marks each and question 3 carries 30 marks.
4. All programming questions must be answered using the C++ language.
5. Write your answers in the Answer Booklet provided.
6. State all assumptions clearly.

Question 1

A quadratic equation takes the form of $y = ax^2 + bx + c$.

Figure Q1-1 illustrates a Unified Modeling Language (UML) class diagram for class **Quadratic**. This class computes the **y** coordinates, displays the equation and writes the resulting coordinates to a file.

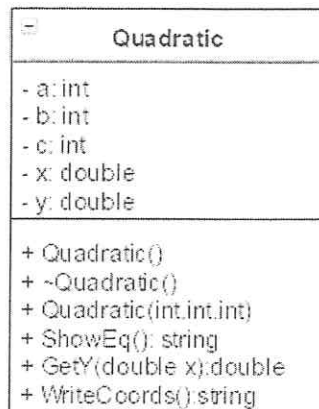


Figure Q1-1: Class **Quadratic** UML diagram.

Using C++ programming language and with reference to Figure Q1-1:

(a) Implement all the member functions, constructor and destructor of class **Quadratic**.

- i. Declare the private members of the class according to the UML diagram in Figure Q1-1. [2 marks]
- ii. The **Quadratic()** default constructor sets the private members to their default values of zero. [1 mark]
- iii. The **Quadratic()** parameterized constructor saves the input **a_in**, **b_in** and **c_in** parameters as quadratic curve coefficients, and initializes the corresponding coefficient member values of the class, or otherwise sets the default values to zero. [3 marks]
- iv. The **~Quadratic()** destructor frees the objects created (if any). [1 mark]
- v. The **ShowEq()** member function returns the quadratic equation as a string using the parameters set by the coefficient private members values. Using the **if** control structure, format the equation accordingly:
 - a) If the coefficient is zero, the term is not displayed.
 - b) If the coefficient is negative, the '+' symbol is not displayed.
 - c) Otherwise display the '+' symbol together with the term.

Figure Q1-2 shows some sample output for **ShowEq()**, using **a=1**, **b=4**,

Continued...

$c=4$, $a=-1$, $b=6$, $c=-8$ and $a=3$, $b=0$, $c=0$. This function throws a string exception "Curve not yet set." if the coefficients have not been set (i.e. throw an exception if **a**, **b** and **c** private member values are 0).

[Hint: use `to_string()` function to convert `int` variables to string values.]

```
Quadratic equation is  $y=x^2+4x+4$ 
Quadratic equation is  $y=-1x^2+6x-8$ 
Quadratic equation is  $y=3x^2$ 
```

Figure Q1-2: sample output from ShowEq()

[7 marks]

- vi. The `GetY()` member function takes in the `x_in` coordinate, sets the `x` member and returns the corresponding `y` coordinate, calculated using the set coefficients. This function throws a string exception "Curve not yet set." if the coefficients have not been set.
- vii. The `WriteCoords()` member function opens the file `curve.txt` to write the quadratic equation of the calling object and the coordinates of the curve using the set coefficients. The function uses the `x` coordinate values of `[-5, 5]` with a resolution of 1 and calls the `GetY()` method function to calculate the corresponding `y` coordinate, then writes the output to `curve.txt`. Close the text file upon write completion. The sample output of the file is given by Figure Q1-3.

[4 marks]

```
Quadratic equation is  $y=-1x^2+6x-8$ 
Coordinates for x=-5 to 5 for equation:
(-5,-63)
(-4,-48)
(-3,-35)
(-2,-24)
(-1,-15)
(0,-8)
(1,-3)
(2,0)
(3,1)
(4,0)
```

Figure Q1-3: curve.txt output.

[6 marks]

- (b) Write a `main()` function to test the implemented class `Quadratic`. Write the `ShowEq()` method for all curve objects within a `try-catch` block. The sample output is displayed in Figure Q1-4.
 - i. Print the header "New Curve($a=1, b=4, c=4$)" and instantiate an object of type class `Quadratic` using $a = 1$, $b = 4$ and $c = 4$ as constructor arguments. Then use the `ShowEq()` method to print out the equation.
 - ii. Print the header "New Curve($a=-1, b=6, c=-8$)" and instantiate an object of type class `Quadratic` using $a = -1$, $b = 6$ and $c = -8$ as constructor arguments. Use the `ShowEq()` method to print out the equation then call `WriteCoords()` to generate the file `curve.txt` as shown in Figure Q1-4.

[4 marks]

Continued...

[4 marks]

- iii. Print the header "New Curve(Unset Curve)" and instantiate an object of type class **Quadratic** without constructor arguments. Use the **ShowEq()** method to print out the equation. The **try-catch** block should display the caught exception.

[3 marks]

```
=====
      New Curve(a=1,b=4,c=4)
=====
Quadratic equation is  $y=x^2+4x+4$ 
One root: X is -2
=====
      New Curve(a=-1,b=6,c=-8)
=====
Quadratic equation is  $y=-1x^2+6x-8$ 
Roots: X is 4 OR 2
=====
      New Curve(unset curve)
=====
Exception occurred: Curve not yet set.
```

Figure Q1-4: Sample output after calling main function.

Note: Use C++ file processing when reading from the text file. Apply the necessary **try** and **catch** procedures in the **main()** function.

Continued...

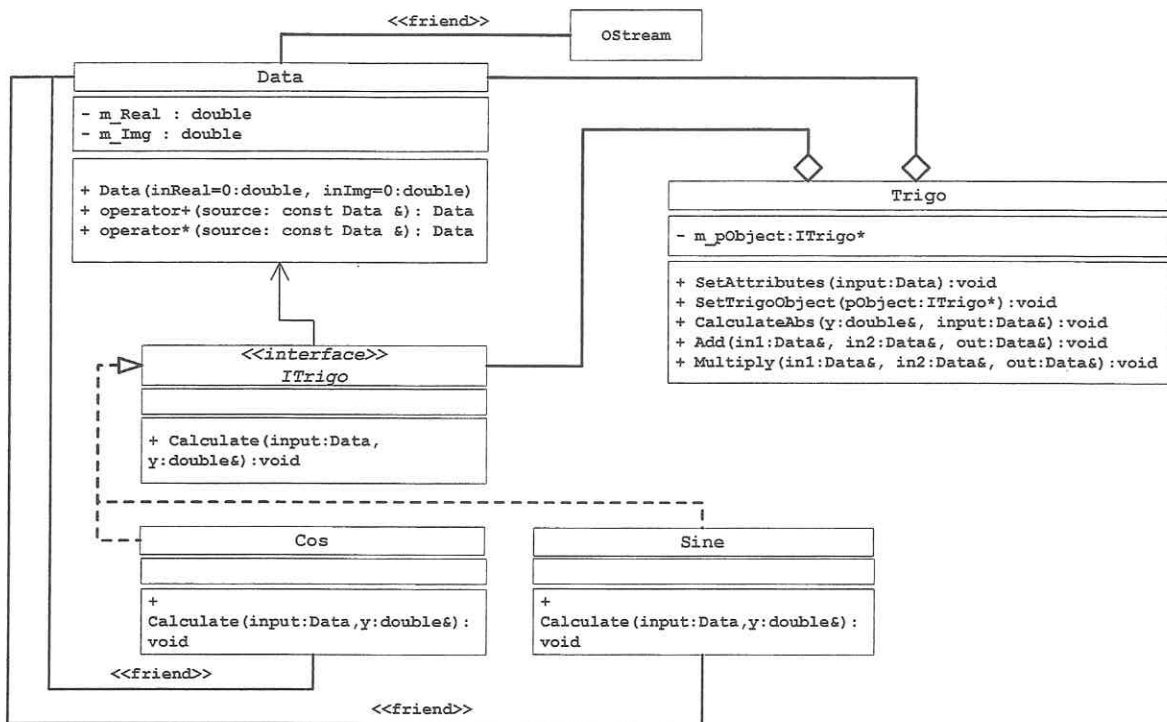
Question 2

Figure Q2: UML class diagram of a trigonometry calculation program for a complex number

Figure Q2 illustrates a **Unified Modeling Language (UML)** class diagram, which describes the design of a trigonometry calculation program for a complex number.

Class **Data** stores the values of the real and imaginary coefficients. These attributes are used to calculate the absolute value of the cosine and sine of a complex number. This class also overloads the addition (+) and multiplication (*) operators.

The function **Calculate()** of classes **Cos** and **Sine** computes and returns **y** (absolute value of the cosines and/or sine of a complex number) as an output argument. Class **Trigo** sets the input values into the member variables and uses the **IPolynomial** class to compute the corresponding polynomial operation. Note that class **ITrigo** is a pure virtual class.

(a) Write complete C++ code, which includes complete class member variables and function definitions (including constructor) for the following classes:

- i. **class Data** (including overloaded output stream operator) [16 marks]
- ii. **class ITrigo** [2 marks]
- iii. **class Cos** [3 marks]
- iv. **class Sine** [3 marks]
- v. **class Trigo** [5 marks]

Continued...

Refer to the class diagram in Figure Q2-1 to assist in identifying the base and implementation classes.

- (b) From part (a), write a **main()** function that will:
- Instantiate two objects of type class **Data** with arbitrary constructor arguments.
[1 mark]
 - Instantiate an object of type class **Trigo**.
[1 mark]
 - Instantiate an **ITrigo** object of type class **Cos** and set this object into the instantiated **Trigo** object.
[2 mark]
 - Test all other member functions in class **Trigo**. Print the output using the overloaded output stream operator of class **Data**. Apply heap memory clean up at the end of the **main()** function.
[2 mark]

The multiplication of a complex number is defined as:

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i$$

The cosine of a complex number is defined as:

$$\cos(a + bi) = \cos a \cosh b - (\sin a \sinh b)i$$

The sine of a complex number is defined as:

$$\sin(a + bi) = \sin a \cosh b + (\cos a \sinh b)i$$

Continued...

Question 3

Figure Q3-1 illustrates a Unified Modeling Language (UML) class diagram for classes **XVar**, **Matrix** and **MatrixOp**. These classes compute the output square of each cell element in a one-dimension heap matrix for a given set of x cell values. All of these classes apply a **class template** design.

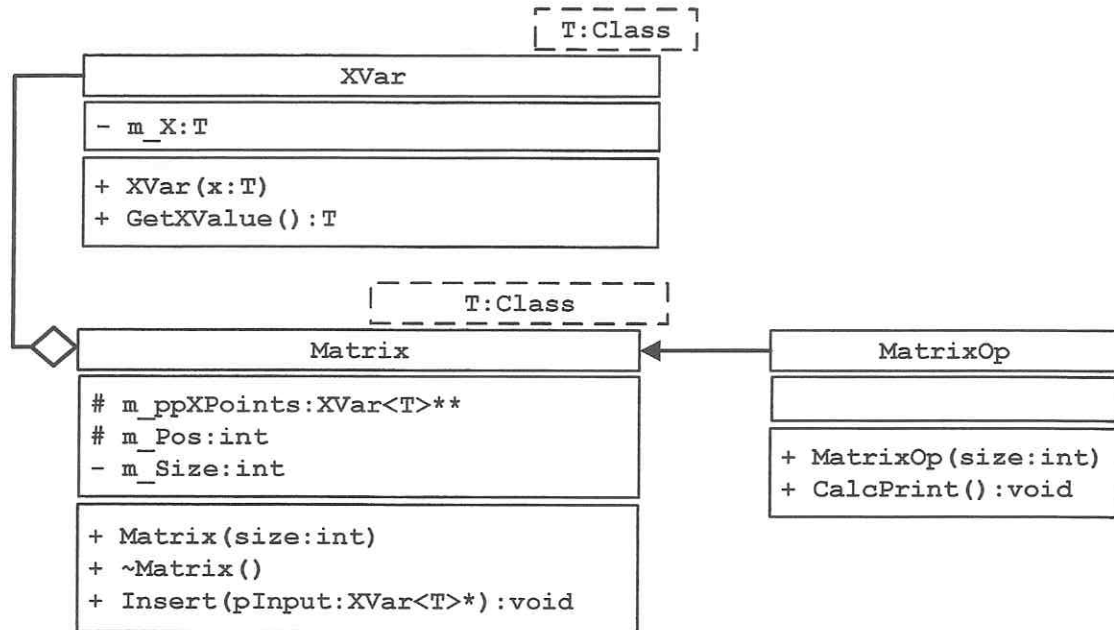


Figure Q3-1: Classes **XVar**, **Matrix**, and **MatrixOp** UML diagram.

- (a) Class **XVar** stores the value of the x variable. This class template accepts one template type specification, which can be defined as **template <class T>**. This allows **XVar** to carry out operations for different data types.

Write complete C++ code for class **XVar** with the following details:

- Proper member variable declaration based on Figure Q3-1. [2 marks]
 - The **XVar()** constructor sets the input constructor argument to the class member variable. [2 marks]
 - The **GetXValue()** method returns the value of the **T** member variable. [3 marks]
- (b) Class **Matrix** stores a heap array of **XVar** type class object based on the input **size** of the constructor argument. The **Matrix** class template accepts one template type specification, which can be defined as **template <class T>**.

Write complete C++ code for class **Matrix** with the following details:

- Proper member variable declarations based on Figure Q3-1. [3 marks]
- The **Matrix()** constructor initializes the **m_Pos** member variable to zero. Note that **m_Pos** keeps track of the number items stored into **m_ppXPoints**. This constructor also creates the heap memory based on the value of the input **size** argument, the starting address of the heap memory is

Continued...

stored into **m_ppXPoints**. The destructor applies the necessary heap memory cleanup.

[5 marks]

- iii. The **Insert()** method inserts the function argument into the **m_ppXPoints** array and increments **m_Pos**. This function throws an exception if the array is full.

[3 marks]

- (c) Class **MatrixOp** inherits from class **Matrix**. Write complete C++ code for class **MatrixOp** with the following details:

- i. Proper class creation and constructor initialization.

[2 marks]

- ii. The **CalcPrint()** method computes and prints the square value for each x cell value in **m_ppXPoints**. Use the **pow()** function from the math library.

[2 marks]

- (d) Write a **main()** function to test the **XVar** and **MatrixOp** classes.

- i. Prompt the user for a size value and instantiate an object of type class **MatrixOp** with **<float>** type.

[3 marks]

- iii. Prompt the user for an x value. Instantiate a heap memory object of type class **XVar** with a **<float>** type. Assign the input x value into this object. Insert the instantiated **XVar** object into the **MatrixOp** object. Apply the necessary **try** and **catch** procedures here.

[3 marks]

- iv. Repeat step (ii) as a loop until user enters a character as an exit value.

[1 mark]

- v. Compute and print out the square of each input x cell value using the **CalcPrint()** member function. Refer to Figure Q3-2 to assist you in developing the **main()** function.

[1 mark]

*Note: Apply the necessary **try** and **catch** procedures in the **main()** function. There is no need for heap memory clean up.*

```
Specify matrix size: 5

Enter a matrix cell value (or a char to exit): 6.8
Enter another matrix cell value (or a char to exit): 7.5
Enter another matrix cell value (or a char to exit): 9.1
Enter another matrix cell value (or a char to exit): a

Square of matrix cells:

Given x = 6.8, Square of 6.8 = 46.24
Given x = 7.5, Square of 7.5 = 56.25
Given x = 9.1, Square of 9.1 = 82.81
```

Figure Q3-2: Sample program execution.

End of Paper